Electronic Transactions on Numerical Analysis.
Volume 31, pp. 68-85, 2008.

ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

A ROBUST AND EFFICIENT PARALLEL SVD SOLVER BASED ON RESTARTED LANCZOS BIDIAGONALIZATION*

VICENTE HERNÁNDEZ†, JOSÉ E. ROMÁN†, AND ANDRÉS TOMÁS†

Abstract. Lanczos bidiagonalization is a competitive method for computing a partial singular value decomposition of a large sparse matrix, that is, when only a subset of the singular values and corresponding singular vectors are required. However, a straightforward implementation of the algorithm has the problem of loss of orthogonality between computed Lanczos vectors, and some reorthogonalization technique must be applied. Also, an effective restarting strategy must be used to prevent excessive growth of the cost of reorthogonalization per iteration. On the other hand, if the method is to be implemented on a distributed-memory parallel computer, then additional precautions are required so that parallel efficiency is maintained as the number of processors increases.

In this paper, we present a Lanczos bidiagonalization procedure implemented in SLEPc, a software library for the solution of large, sparse eigenvalue problems on parallel computers. The solver is numerically robust and scales well up to hundreds of processors.

Key words. Partial singular value decomposition, Lanczos bidiagonalization, thick restart, parallel computing.

AMS subject classifications. 65F15, 15A18, 65F50.

1. Introduction. The computation of singular subspaces associated with the k largest or smallest singular values of a large, sparse (or structured) matrix A is commonplace. Example applications are the solution of discrete ill-posed problems [17], or the construction of low-rank matrix approximations in areas such as signal processing [13] and information retrieval [5]. This paper focuses on Lanczos bidiagonalization, a method that can be competitive in this context because it exploits matrix sparsity.

The problem of computing the singular value decomposition (SVD) of a matrix A can be formulated as an equivalent eigenvalue problem, using for instance the cross product matrix A^*A. The Lanczos bidiagonalization algorithm can be deduced from Lanczos tridiagonalization applied to these equivalent eigenproblems. Therefore, it inherits the good properties as well as the implementation difficulties present in Lanczos-based eigensolvers. It is possible to stop after a few Lanczos steps, in which case we obtain Rayleigh-Ritz approximations of the singular triplets. On the other hand, loss of orthogonality among Lanczos vectors has to be dealt with, either by full reorthogonalization or by a cheaper alternative, such as partial reorthogonalization [25, 26]. Block variants of the method have been proposed; see, e.g., [16]. Also, in the case of slow convergence, restarting techniques become very important in order to keep the cost of reorthogonalization bounded. All these techniques are intended for numerical robustness as well as computational efficiency. Furthermore, if these properties are to be maintained in the context of parallel computing, then additional tuning of the algorithm may be required. Therefore, it becomes apparent that implementing an industrial-strength SVD solver based on Lanczos bidiagonalization requires a careful combination of a number of different techniques.

In this paper, we present a thick restart Lanczos bidiagonalization procedure implemented in SLEPc, the Scalable Library for Eigenvalue Problem Computations [19, 20].

*Received November 30, 2007. Accepted July 18, 2008. Published online on January 19, 2009. Recommended by Daniel Kressner.

†Instituto ITACA, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, ({vhernand, jroman, antodo}@itaca.upv.es). Work partially supported by the Valencia Regional Administration, Directorate of Research and Technology Transfer, under grant number GV06/091, by Universidad Politécnica de Valencia, under program number PAID-04-07, and by the Centro para el Desarrollo Tecnologico Industrial (CDTI) of the Spanish Ministry of Industry, Tourism and Commerce through the CDTEAM Project (Consortium for the Development of Advanced Medicine Technologies).

ETNA
Kent State University
http://etna.math.kent.edu

The proposed Lanczos bidiagonalization algorithm is based on full reorthogonalization via iterated Classical Gram-Schmidt, and its main goal is to reduce the number of synchronization points in the parallel implementation, while maintaining numerical robustness and fast convergence. Some of the techniques presented here were also applied to the Arnoldi eigensolver in a previous work by the authors [18]. The implemented bidiagonalization algorithm is used as a basis for a thick-restarted SVD solver similar to that proposed by Baglama and Reichel [1].

The text is organized as follows. First, Sections 2-4 provide a general description of the Lanczos bidiagonalization method, discuss how to deal with loss of orthogonality among Lanczos vectors, and review the thick-restarted strategy for singular value solvers. Then, Section 5 gives some details about the SLEPc implementation. Finally, Sections 6 and 7 show some numerical and performance results obtained with this implementation.

2. Lanczos bidiagonalization. The singular value decomposition of an $m \times n$ complex matrix A can be written as

$$A = U\Sigma V^*, \tag{2.1}$$

where $U = [u_1, \ldots, u_m]$ is an $m \times m$ unitary matrix ($U^*U = I$), $V = [v_1, \ldots, v_n]$ is an $n \times n$ unitary matrix ($V^*V = I$), and Σ is an $m \times n$ diagonal matrix with nonnegative real diagonal entries $\Sigma_{ii} = \sigma_i$, for $i = 1, \ldots, \min\{m, n\}$. If A is real, U and V are real and orthogonal. The vectors u_i are called the left singular vectors, the v_i are the right singular vectors, and the σ_i are the singular values. In this work, we will assume without loss of generality that $m \geq n$. The singular values are labeled in descending order, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$.

The problem of computing the singular triplets (σ_i, u_i, v_i) of A can be formulated as an eigenvalue problem involving a Hermitian matrix related to A, either

1. the *cross product* matrix, A^*A, or
2. the *cyclic* matrix, $H(A) = \begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$.

The singular values are the nonnegative square roots of the eigenvalues of the cross product matrix. This approach may imply a severe loss of accuracy in the smallest singular values. The cyclic matrix approach is an alternative procedure that avoids this problem, at the expense of significantly increasing the cost of the computation. Note that we could also consider the alternative cross product matrix AA^*, but that approach is unfeasible under the assumption that $m \geq n$.

Computing the cross product matrix explicitly is not recommended, especially in the case of A sparse. Bidiagonalization was proposed by Golub and Kahan [15] as a way of tridiagonalizing the cross product matrix without forming it explicitly. Consider the decomposition

$$A = PBQ^*, \tag{2.2}$$

where P and Q are unitary matrices, and B is an $m \times n$ upper bidiagonal matrix. Then the tridiagonal matrix B^*B is unitarily similar to A^*A. Additionally, specific methods exist (e.g., [11]) that compute the singular values of B without forming B^*B. Therefore, after computing the SVD of B,

$$B = X\Sigma Y^*, \tag{2.3}$$

it only remains to combine (2.3) and (2.2) to get the solution of the original problem (2.1) with $U = PX$ and $V = QY$.

Bidiagonalization can be accomplished by means of Householder transformations or alternatively via Lanczos recurrences. The latter approach is more appropriate for sparse matrix

ETNA
Kent State University
http://etna.math.kent.edu

computations and was already proposed in [15], hence it is sometimes referred to as Golub-Kahan-Lanczos bidiagonalization.

The Lanczos bidiagonalization technique can be derived from several equivalent perspectives. Consider a compact version of (2.2),

$$A = P_n B_n Q_n^*,$$

where the zero rows of the bidiagonal matrix have been removed and, therefore, P_n is now an $m \times n$ matrix with orthonormal columns, Q_n is a unitary matrix of order n that is equal to Q of (2.2), and B_n is a square matrix of order n that can be written as

$$B_n = P_n^* A Q_n = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \alpha_3 & \beta_3 & & \\ & & & \ddots & \ddots & \\ & & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & & \alpha_n \end{bmatrix}. \tag{2.4}$$

The coefficients of this matrix are real and given by $\alpha_j = p_j^* A q_j$ and $\beta_j = p_j^* A q_{j+1}$, where p_j and q_j are the columns of P_n and Q_n, respectively. It is possible to derive a double recurrence to compute these coefficients together with the vectors p_j and q_j, since after choosing q_1 as an arbitrary unit vector, the other columns of P_n and Q_n are determined uniquely (apart from a sign change, and assuming A has full rank and B_n is unreduced).

Pre-multiplying (2.4) by P_n, we have the relation $AQ_n = P_n B_n$. Also, if we transpose both sides of (2.4) and pre-multiply by Q_n, we obtain $A^* P_n = Q_n B_n^*$. Equating the first $k < n$ columns of both relations results in

$$AQ_k = P_k B_k, \tag{2.5}$$
$$A^* P_k = Q_k B_k^* + \beta_k q_{k+1} e_k^*, \tag{2.6}$$

where B_k denotes the $k \times k$ leading principal submatrix of B_n. Analogous expressions can be written in vector form by equating the jth column only,

$$\begin{aligned} Aq_j &= \beta_{j-1} p_{j-1} + \alpha_j p_j, \\ A^* p_j &= \alpha_j q_j + \beta_j q_{j+1}. \end{aligned} \tag{2.7}$$

These expressions directly yield the double recursion

$$\alpha_j p_j = Aq_j - \beta_{j-1} p_{j-1}, \tag{2.8}$$
$$\beta_j q_{j+1} = A^* p_j - \alpha_j q_j, \tag{2.9}$$

with $\alpha_j = \|Aq_j - \beta_{j-1} p_{j-1}\|_2$ and $\beta_j = \|A^* p_j - \alpha_j q_j\|_2$, since the columns of P_n and Q_n are normalized. The bidiagonalization algorithm is built from (2.8) and (2.9); see Algorithm 1.

Equations (2.5) and (2.6) can be combined by pre-multiplying the first one by A^*, resulting in

$$A^* A Q_k = Q_k B_k^* B_k + \alpha_k \beta_k q_{k+1} e_k^*. \tag{2.10}$$

The matrix $B_k^* B_k$ is symmetric positive definite and tridiagonal. The conclusion is that Algorithm 1 computes the same information as the Lanczos tridiagonalization algorithm applied

ALGORITHM 1 (Golub-Kahan-Lanczos Bidiagonalization).
Choose a unit-norm vector q_1
Set $\beta_0 = 0$
For $j = 1, 2, \ldots, k$
$p_j = Aq_j - \beta_{j-1}p_{j-1}$
$\alpha_j = \|p_j\|_2$
$p_j = p_j/\alpha_j$
$q_{j+1} = A^* p_j - \alpha_j q_j$
$\beta_j = \|q_{j+1}\|_2$
$q_{j+1} = q_{j+1}/\beta_j$
end

---

to the Hermitian matrix A^*A. In particular, the right Lanczos vectors q_j computed by Algorithm 1 constitute an orthonormal basis of the Krylov subspace

$$\mathcal{K}_k(A^*A, q_1) = \operatorname{span}\{q_1, A^*Aq_1, \ldots, (A^*A)^{k-1}q_1\}.$$

Another way of combining (2.5) and (2.6) is by pre-multiplying the second one by A, giving in this case the equation

$$AA^*P_k = P_kB_kB_k^* + \beta_kAq_{k+1}e_k^*.$$

In contrast to (2.10), this equation does not represent a Lanczos decomposition, because the vector Aq_{k+1} is not orthogonal to P_k, in general. However, using (2.7) we get

$$\begin{aligned} AA^*P_k &= P_kB_kB_k^* + \beta_k^2 p_k e_k^* + \beta_k\alpha_{k+1}p_{k+1}e_k^* \\ &= P_k(B_kB_k^* + \beta_k^2 e_k e_k^*) + \beta_k\alpha_{k+1}p_{k+1}e_k^*, \end{aligned}$$

where the matrix $B_kB_k^* + \beta_k^2 e_k e_k^*$ is also symmetric positive definite and tridiagonal. Thus, a similar conclusion can be drawn for matrix AA^*, and the left Lanczos vectors p_j span the Krylov subspace $\mathcal{K}_k(AA^*, p_1)$.

There is an alternative way of deriving Algorithm 1, which further displays the intimate relation between Lanczos bidiagonalization and the usual three-term Lanczos tridiagonalization. The idea is to apply the standard Lanczos algorithm to the cyclic matrix, $H(A)$, with the special initial vector

$$z_1 = \begin{bmatrix} 0 \\ q_1 \end{bmatrix}.$$

It can be shown that the generated Lanczos vectors are then

$$z_{2j-1} = \begin{bmatrix} 0 \\ q_j \end{bmatrix} \quad \text{and} \quad z_{2j} = \begin{bmatrix} p_j \\ 0 \end{bmatrix}, \tag{2.11}$$

and that the projected matrix after $2k$ Lanczos steps is

$$T_{2k} = \begin{bmatrix} 0 & \alpha_1 & & & & & \\ \alpha_1 & 0 & \beta_1 & & & & \\ & \beta_1 & 0 & \alpha_2 & & & \\ & & \alpha_2 & 0 & \beta_2 & & \\ & & & \beta_2 & 0 & \ddots & \\ & & & & \ddots & \ddots & \alpha_k \\ & & & & & \alpha_k & 0 \end{bmatrix}.$$

ETNA
Kent State University
http://etna.math.kent.edu

That is, two steps of this procedure compute the same information as one step of Algorithm 1. Moreover, it is easy to show that there is an equivalence transformation with the odd-even permutation (also called perfect shuffle) that maps T_{2k} into the cyclic matrix $H(B_k)$. Note that, in a computer implementation, this procedure would require about twice as much storage as Algorithm 1, unless the zero components in the Lanczos vectors (2.11) are not stored explicitly.

Due to these equivalences, all the properties and implementation considerations of Lanczos tridiagonalization (see [3], [24], [27] or [29]) carry over to Algorithm 1. In particular, error bounds for Ritz approximations can be computed very easily. After k Lanczos steps, the Ritz values $\tilde{\sigma}_i$ (approximate singular values of A) are computed as the singular values of B_k, and the Ritz vectors are

$$\tilde{u}_i = P_k x_i, \qquad \tilde{v}_i = Q_k y_i,$$

where x_i and y_i are the left and right singular vectors of B_k. With these definitions, and equations (2.5)-(2.6), it is easy to show that

$$A\tilde{v}_i = \tilde{\sigma}_i \tilde{u}_i, \qquad A^*\tilde{u}_i = \tilde{\sigma}_i \tilde{v}_i + \beta_k q_{k+1} e_k^* x_i.$$

The residual norm associated to the Ritz singular triplet $(\tilde{\sigma}_i, \tilde{u}_i, \tilde{v}_i)$, defined as

$$\|r_i\|_2 = \left(\|A\tilde{v}_i - \tilde{\sigma}_i \tilde{u}_i\|_2^2 + \|A^*\tilde{u}_i - \tilde{\sigma}_i \tilde{v}_i\|_2^2 \right)^{\frac{1}{2}},$$

can be cheaply computed as

$$\|r_i\|_2 = \beta_k |e_k^* x_i|. \tag{2.12}$$

3. Dealing with loss of orthogonality. As in the case of the standard Lanczos tridiagonalization algorithm, Algorithm 1 diverts from the expected behaviour when run in finite precision arithmetic. In particular, after a sufficient number of steps the Lanczos vectors start to lose their mutual orthogonality, and this happens together with the appearance of repeated and spurious Ritz values in the set of singular values of B_j.

The simplest cure for this loss of orthogonality is full orthogonalization. In Lanczos bidiagonalization, two sets of Lanczos vectors are computed, so full orthogonalization amounts to orthogonalizing vector p_j explicitly with respect to all the previously computed left Lanczos vectors, and orthogonalizing vector q_{j+1} explicitly with respect to all the previously computed right Lanczos vectors. Algorithm 2 shows this variant with a modified Gram-Schmidt (MGS) orthogonalization procedure. Note that in the computation of p_j it is no longer necessary to subtract the term $\beta_{j-1} p_{j-1}$, since this is already done in the orthogonalization step; a similar remark holds for the computation of q_{j+1}.

This solution was already proposed in the seminal paper by Golub and Kahan [15], and used in some of the first implementations, such as the block version in [16]. The main advantage of full orthogonalization is its robustness, since orthogonality is maintained to full machine precision (provided that reorthogonalization is employed, see Section 5 for details). Its main drawback is the high computational cost, which grows as the iteration proceeds.

An alternative to full orthogonalization is to simply ignore loss of orthogonality and perform only local orthogonalization at every Lanczos step. This technique has to carry out a post-process of matrix T_{2k} in order to determine the correct multiplicity of the computed singular values as well as to discard the spurious ones; see [8] for further details.

Semiorthogonal techniques try to find a compromise between full and local orthogonalization. One such technique is partial reorthogonalization [30], which uses a cheap recurrence

ETNA
Kent State University
http://etna.math.kent.edu

ALGORITHM 2 (Lanczos Bidiagonalization with Full Orthogonalization).

Choose a unit-norm vector q_1
For $j = 1, 2, \ldots, k$
    $p_j = Aq_j$
    For $i = 1, 2, \ldots, j-1$
        $\gamma = p_i^* p_j$
        $p_j = p_j - \gamma p_i$
    end
    $\alpha_j = \|p_j\|_2$
    $p_j = p_j / \alpha_j$
    $q_{j+1} = A^* p_j$
    For $i = 1, 2, \ldots, j$
        $\gamma = q_i^* q_{j+1}$
        $q_{j+1} = q_{j+1} - \gamma q_i$
    end
    $\beta_j = \|q_{j+1}\|_2$
    $q_{j+1} = q_{j+1} / \beta_j$
end

---

to estimate the level of orthogonality, and applies some corrective measures when it drops below a certain threshold. This technique has been adapted by Larsen [25] to the particular case of Lanczos bidiagonalization. In this case, two recurrences are necessary, one for monitoring loss of orthogonality among right Lanczos vectors, and the other one for left Lanczos vectors.

However, these alternatives to full orthogonalization are not very meaningful in the context of restarted variants, discussed in Section 4. First, the basis size is limited so the cost of full orthogonalization does not grow indefinitely. Second, currently there is no reliable theory background on how to adapt semiorthogonal techniques to the case in which a restart is performed. In [7] an attempt is done to extend semi-orthogonalization also to locked eigenvectors in the context of an explicitly restarted eigensolver. In the case of thick restart, the technique employed in this paper, numerical experiments carried out by the authors show that orthogonality with respect to restart vectors must be enforced explicitly in each iteration, negating the advantage of semiorthogonal techniques.

One-sided variant. There is a variation of Algorithm 2 that maintains the effectiveness of full reorthogonalization, but with a considerably reduced cost. This technique was proposed by Simon and Zha [31]. The idea comes from the observation that, in the Lanczos bidiagonalization procedure without reorthogonalization, the level of orthogonality of left and right Lanczos vectors go hand in hand. If we quantify the level of orthogonality of the Lanczos vectors $\hat{P}_j$ and $\hat{Q}_j$, computed in finite precision arithmetic, as

$$\eta(\hat{P}_j) = \|I - \hat{P}_j^* \hat{P}_j\|_2, \quad \eta(\hat{Q}_j) = \|I - \hat{Q}_j^* \hat{Q}_j\|_2,$$

then it can be observed that at a given Lanczos step j, $\eta(\hat{P}_j)$ and $\eta(\hat{Q}_j)$ differ in no more than an order of magnitude, except maybe when B_j becomes very ill-conditioned. This observation led Simon and Zha to propose what they called the one-sided version, shown in Algorithm 3.

Note that the only difference of Algorithm 3 with respect to Algorithm 2 is that p_j is no longer orthogonalized explicitly. Still, numerical experiments carried out by Simon and Zha show that the computed $\hat{P}_j$ vectors maintain a similar level of orthogonality as $\hat{Q}_j$.

ETNA
Kent State University
http://etna.math.kent.edu

ALGORITHM 3 (One-Sided Lanczos Bidiagonalization).

Choose a unit-norm vector q_1
Set $\beta_0 = 0$
For $j = 1, 2, \ldots, k$
    $p_j = Aq_j - \beta_{j-1} p_{j-1}$
    $\alpha_j = \|p_j\|_2$
    $p_j = p_j / \alpha_j$
    $q_{j+1} = A^* p_j$
    For $i = 1, 2, \ldots, j$
        $\gamma = q_i^* q_{j+1}$
        $q_{j+1} = q_{j+1} - \gamma q_i$
    end
    $\beta_j = \|q_{j+1}\|_2$
    $q_{j+1} = q_{j+1} / \beta_j$
end

When the singular values of interest are the smallest ones, then B_j may become ill-conditioned. A robust implementation should track this event and revert to the two-sided variant when a certain threshold is exceeded.

4. Restarted bidiagonalization. Restarting is a key aspect in the efficient implementation of projection-based eigensolvers, such as those based on Arnoldi or Jacobi-Davidson. This topic has motivated an intense research activity in recent years. These developments are also applicable to Lanczos, especially if full reorthogonalization is employed. In this section, we adapt the discussion to the context of Lanczos bidiagonalization.

The number of iterations required in the Lanczos bidiagonalization algorithm (i.e., the value of k) can be quite high if many singular triplets are requested, and also depends on the distribution of the singular values, as convergence is slow in the presence of clustered singular values. Increasing k too much may not be acceptable, since this implies a growth in storage requirements and, sometimes more important, a growth of computational cost per iteration in the case of full orthogonalization. To avoid this problem, restarted variants limit the maximum number of Lanczos steps to a fixed value k, and when this value is reached the computation is re-initiated. This can be done in different ways.

Explicit restart consists of rerunning the algorithm with a "better" initial vector. In Algorithms 1-3, the initial vector is q_1, so the easiest strategy is to replace q_1 with the right Ritz vector associated to the approximate dominant singular value. A block equivalent of this technique was employed in [16]. In the case that many singular triplets are to be computed, it is not evident how to build the new q_1. One possibility is to compute q_1 as a linear combination of a subset of the computed Ritz vectors, possibly applying a polynomial filter to remove components in unwanted directions.

Implicit restart is a much better alternative that eliminates the need to explicitly compute a new start vector q_1. It consists of combining the Lanczos bidiagonalization process with the implicitly shifted QR algorithm. The k-step Lanczos relations described in (2.5)-(2.6) are transformed and truncated to order $\ell < k$, and then extended again to order k. The procedure allows the small-size equations to retain the relevant spectral information of the full-size relations. A detailed description of this technique can be found in [6], [21], [23] and [26].

An equivalent yet easier to implement alternative to implicit restart is the so-called thick restart, originally proposed in the context of Lanczos tridiagonalization [32]. We next de-

ETNA
Kent State University
http://etna.math.kent.edu

scribe how this method can be adapted to Lanczos bidiagonalization, as proposed in [1].

The main idea of thick-restarted Lanczos bidiagonalization is to reduce the full k-step Lanczos bidiagonalization, (2.5)-(2.6), to the following one

$$A\tilde{Q}_{\ell+1} = \tilde{P}_{\ell+1}\tilde{B}_{\ell+1}, \tag{4.1}$$
$$A^*\tilde{P}_{\ell+1} = \tilde{Q}_{\ell+1}\tilde{B}^*_{\ell+1} + \tilde{\beta}_{\ell+1}\tilde{q}_{\ell+2}e^*_{k+1}, \tag{4.2}$$

where the value of $\ell < k$ could be, for instance, the number of wanted singular values. The key point here is to build the decomposition of (4.1)-(4.2) in such a way that it keeps the relevant spectral information contained in the full decomposition. This is achieved directly by setting the first ℓ columns of $\tilde{Q}_{\ell+1}$ to be the wanted approximate right singular vectors, and analogously in $\tilde{P}_{\ell+1}$ the corresponding approximate left singular vectors. It can be shown [1] that it is possible to easily build a decomposition that satisfies these requirements, as described in the following.

We start by defining $\tilde{Q}_{\ell+1}$ as

$$\tilde{Q}_{\ell+1} = \left[\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_\ell, q_{k+1}\right], \tag{4.3}$$

that is, the Ritz vectors $\tilde{v}_i = Q_k y_i$ together with the last Lanczos vector generated by Algorithm 1. Note that this matrix has orthonormal columns because $Q_k^* q_{k+1} = 0$ by construction. Similarly, define $\tilde{P}_{\ell+1}$ as

$$\tilde{P}_{\ell+1} = \left[\tilde{u}_1, \tilde{u}_2, \ldots, \tilde{u}_\ell, \tilde{p}_{\ell+1}\right], \tag{4.4}$$

with $\tilde{u}_i = P_k x_i$, and $\tilde{p}_{\ell+1}$ a unit-norm vector computed as $\tilde{p}_{\ell+1} = f/\|f\|_2$, where f is the vector resulting from orthogonalizing Aq_{k+1} with respect to the first ℓ left Ritz vectors, $\tilde{u}_i$,

$$f = Aq_{k+1} - \sum_{i=1}^{\ell} \tilde{\rho}_i \tilde{u}_i.$$

It can be shown that the orthogonalization coefficients can be computed as $\tilde{\rho}_i = \beta_k e_k^* x_i$; note that these values are similar to the residual bounds in (2.12), but here the sign is relevant. The new projected matrix is

$$\tilde{B}_{\ell+1} = \begin{bmatrix} \tilde{\sigma}_1 & & & & \tilde{\rho}_1 \\ & \tilde{\sigma}_2 & & & \tilde{\rho}_2 \\ & & \ddots & & \vdots \\ & & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell \\ & & & & \tilde{\alpha}_{\ell+1} \end{bmatrix},$$

where $\tilde{\alpha}_{\ell+1} = \|f\|_2$, so that (4.1) holds. To complete the form of a Lanczos bidiagonalization, it only remains to define $\tilde{\beta}_{\ell+1}$ and $\tilde{q}_{\ell+2}$ in (4.2), which turn out to be $\tilde{\beta}_{\ell+1} = \|g\|_2$ and $\tilde{q}_{\ell+2} = g/\|g\|_2$, where $g = A^*\tilde{p}_{\ell+1} - \tilde{\alpha}_{\ell+1}q_{k+1}$.

It is shown in [1] that the Lanczos bidiagonalization relation is maintained if Algorithm 1 is run for $j = \ell+2, \ldots, k$, starting from the values of $\tilde{\beta}_{\ell+1}$ and $\tilde{q}_{\ell+2}$ indicated above, thus obtaining a new full-size decomposition. In this case, the projected matrix is no longer

ETNA
Kent State University
http://etna.math.kent.edu

ALGORITHM 4 (Thick-restart Lanczos Bidiagonalization).
Input: Matrix A, initial unit-norm vector q_1, and number of steps k
Output: $\ell \leq k$ Ritz triplets
1. Build an initial Lanczos bidiagonalization of order k
2. Compute Ritz approximations of the singular triplets
3. Truncate to a Lanczos bidiagonalization of order ℓ
4. Extend to a Lanczos bidiagonalization of order k
5. Check convergence and if not satisfied, go to step 2

---

bidiagonal, but it takes the form

$$\tilde{B}_k = \begin{bmatrix} \tilde{\sigma}_1 & & & & \tilde{\rho}_1 & & & \\ & \tilde{\sigma}_2 & & & \tilde{\rho}_2 & & & \\ & & \ddots & & \vdots & & & \\ & & & \tilde{\sigma}_\ell & \tilde{\rho}_\ell & & & \\ & & & & \tilde{\alpha}_{\ell+1} & \beta_{\ell+1} & & \\ & & & & & \ddots & \ddots & \\ & & & & & & \alpha_{k-1} & \beta_{k-1} \\ & & & & & & & \alpha_k \end{bmatrix},$$

where the values without tilde are computed in the usual way with Algorithm 1.

When carried out in an iterative fashion, the above procedure results in Algorithm 4. The step 4 can be executed by a variation of Algorithm 3, as illustrated in Algorithm 5. Starting from the new initial vector, $q_{\ell+1}$, this algorithm first computes the corresponding left initial vector $p_{\ell+1}$, and then proceeds in the standard way.

ALGORITHM 5 (One-Sided Lanczos Bidiagonalization – restarted).
$p_{\ell+1} = Aq_{\ell+1}$
For $i = 1, 2, \ldots, \ell$
  $p_{\ell+1} = p_{\ell+1} - \tilde{\rho}_i p_i$
end
For $j = \ell + 1, \ell + 2, \ldots, k$
  $\alpha_j = \|p_j\|_2$
  $p_j = p_j / \alpha_j$
  $q_{j+1} = A^* p_j$
  For $i = 1, 2, \ldots, j$
    $\gamma = q_i^* q_{j+1}$
    $q_{j+1} = q_{j+1} - \gamma q_i$
  end
  $\beta_j = \|q_{j+1}\|_2$
  $q_{j+1} = q_{j+1} / \beta_j$
  If $j < k$
    $p_{j+1} = Aq_{j+1} - \beta_j p_j$
  end
end

ETNA
Kent State University
http://etna.math.kent.edu

5. Parallel implementation in SLEPc. This work is framed in the context of the SLEPc project. SLEPc, the Scalable Library for Eigenvalue Problem Computations [19, 20], is a parallel software library intended for the solution of large, sparse eigenvalue problems. A collection of eigensolvers is provided, that can be used for different types of eigenproblems, either standard or generalized ones, both Hermitian and non-Hermitian, with either real or complex arithmetic. SLEPc also provides built-in support for spectral transformations such as shift-and-invert. In addition, a mechanism for computing partial singular value decompositions is also available, either via the associated eigenproblems (cross product or cyclic matrix) or with Lanczos bidiagonalization as described in this paper.

SLEPc is an extension of PETSc, the Portable, Extensible Toolkit for Scientific Computation [4], and therefore reuses part of its software infrastructure, particularly the matrix and vector data structures. PETSc uses a message-passing programming paradigm with standard data distribution of vectors and matrices, i.e., by blocks of rows. With this distribution of data across processors, parallelization of the Lanczos bidiagonalization process (Algorithm 5) amounts to carrying out the following three stages in parallel:

1. Basis expansion. In the case of Lanczos bidiagonalization, the method builds two bases, one associated to A and another to A^*, so this stage consists of a direct and transpose sparse matrix-vector product. In many applications, the matrix-vector product operation can be parallelized quite efficiently. This is the case in mesh-based computations, in which, if the mesh is correctly partitioned in compact subdomains, data needs to be exchanged only between processes owning neighbouring subdomains.
2. Orthogonalization. This stage consists of a series of vector operations such as inner products, additions, and multiplications by a scalar.
3. Normalization. From the parallelization viewpoint, the computation of the norm is equivalent to a vector inner product.

The matrix vector products at the core of the Lanczos process are implemented with PETSc's MatMult and MatMultTranspose operations. These operations are optimized for parallel sparse storage and even allow for matrix-free, user-defined matrix-vector product operations; see [4] for additional details. PETSc matrices are stored in compressed sparse row (CSR) format so the MatMult operation achieves good cache memory efficiency. This operation is implemented as a loop that traverses the non-zero matrix elements and stores the resulting vector elements in order. However, the MatMultTranspose operation writes the resulting vector elements in a non-consecutive order, forcing frequent cache block copies to main memory. This difference in performance is evident especially in the case of rectangular matrices. In order to achieve good sequential efficiency, SLEPc stores the matrix A^* explicitly. This detail is transparent to the user and it can be deactivated to reduce memory usage. We will center our discussion on the orthogonalization and normalization stages, and assume that the implementation of the basis expansion is scalable. The test cases used for the performance analysis presented in Section 7 have been chosen so that basis expansion has a negligible impact on scalability. Also, in those matrices the decision on the explicit storage of A^* makes little difference.

Vector addition and scaling operations can be parallelized trivially, with no associated communication. Therefore, the parallel efficiency of the orthogonalization and normalization steps only depends on how the required inner products are performed. The parallelization of a vector inner product requires a global reduction operation (an all-reduce addition), which on distributed-memory platforms has a significant cost, growing with the number of processes. Moreover, this operation represents a global synchronization point in the algorithm, thus hampering scalability if done too often. Consequently, global reduction operations should be eliminated whenever possible, for instance by grouping together several inner products

ETNA
Kent State University
http://etna.math.kent.edu

in a single reduction. The multiple-vector product operation accomplishes all the individual inner products with just one synchronization point and roughly the same communication cost as just one inner product. A detailed analysis of these techniques applied to the Arnoldi method for eigenproblems has been published by the authors [18].

As a consequence, classical Gram-Schmidt (CGS) orthogonalization scales better than the MGS procedure used in Algorithm 5, because CGS computes all the required inner products with the unmodified vector q_{j+1}, that is, $c = Q_j^* q_{j+1}$, and this operation can be carried out with a single communication. However, CGS is known to be numerical unstable when implemented in finite precision arithmetic. This problem can be solved by iterating the CGS procedure, until the resulting vector is sufficiently orthogonal to working accuracy. We will refer to this technique as selective reorthogonalization. A simple criterion can be used to avoid unnecessary reorthogonalization steps [9]. This criterion involves the computation of the vector norms before and after the orthogonalization. The parallel overhead associated to the first vector norm can be eliminated by joining its communication with the inner products corresponding to the computation of the orthogonalization coefficients c. The explicit computation of the second norm can be avoided with a simple technique used in [14]. The basic idea is to estimate this norm starting from the original norm (before the orthogonalization), by simply applying the Pythagorean theorem as described later in this section. Usually, at most one reorthogonalization is necessary in practice, although provision has to be made for a second one to handle specially ill-conditioned cases.

Another parallel optimization that can be applied to Algorithm 5 is to postpone the normalization of p_j until after the orthogonalization of q_{j+1}. This allows for the union of two global communications, thus eliminating one synchronization point. As a side effect, the basis expansion has to be done with an unnormalized vector, but this is not a problem provided that all the computed quantities are corrected as soon as the norm is available. This technique was originally proposed in [22] in the context of an Arnoldi eigensolver without reorthogonalization.

Applying all the above mentioned optimizations to Algorithm 5 results in Algorithm 6. In this algorithm, lines 7 and 14 are the CGS orthogonalization step, and lines 17 and 19 are the CGS reorthogonalization step. The selective reorthogonalization criterion is checked in line 16, typically with a value of $\eta = 1/\sqrt{2}$ as suggested in [9, 28]. In this expression, ρ represents the norm of q_{j+1} before the orthogonalization. This value is computed explicitly, as discussed later in this section. In contrast, the norm after orthogonalization, β_j, is estimated in lines 15 and 20. These estimations are based on the following relation due to lines 14 and 19,

$$q_{j+1} = q'_{j+1} - Q_j c \,, \tag{5.1}$$

where q_{j+1} and q'_{j+1} denote the vector before and after reorthogonalization, respectively, and $Q_j c$ is the vector resulting from projecting q_{j+1} onto $\operatorname{span}\{q_1, q_2, \ldots, q_j\}$. In exact arithmetic, q'_{j+1} is orthogonal to this subspace and it is possible to apply the Pythagorean theorem to the right-angled triangle formed by these three vectors

$$\|q_{j+1}\|_2^2 = \|q'_{j+1}\|_2^2 + \|Q_j c\|_2^2 \,. \tag{5.2}$$

Since the columns of Q_j are orthonormal, the wanted norm can be computed as

$$\|q'_{j+1}\|_2 = \sqrt{\|q_{j+1}\|_2^2 - \textstyle\sum_{i=1}^{j} c_i^2} \,. \tag{5.3}$$

In deriving (5.3), we have assumed that q'_{j+1} is orthogonal to $\operatorname{span}\{q_1, q_2, \ldots, q_j\}$ and that the columns of Q_j are orthonormal. These assumptions are not necessarily satisfied in

ETNA
Kent State University
http://etna.math.kent.edu

ALGORITHM 6 (One-Sided Lanczos Bidiag. – restarted, with enhancements).

1 $p_{\ell+1} = Aq_{\ell+1}$
2 For $i = 1, 2, \ldots, \ell$
3 $\quad p_{\ell+1} = p_{\ell+1} - \tilde{\rho}_i p_i$
4 end
5 For $j = \ell+1, \ell+2, \ldots, k$
6 $\quad q_{j+1} = A^* p_j$
7 $\quad c = Q_j^* q_{j+1}$
8 $\quad \rho = \|q_{j+1}\|_2$
9 $\quad \alpha_j = \|p_j\|_2$
10 $\quad p_j = p_j / \alpha_j$
11 $\quad q_{j+1} = q_{j+1} / \alpha_j$
12 $\quad c = c / \alpha_j$
13 $\quad \rho = \rho / \alpha_j$
14 $\quad q_{j+1} = q_{j+1} - Q_j c$
15 $\quad \beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_i^2}$
16 $\quad$ If $\beta_j < \eta\rho$
17 $\quad\quad c = Q_j^* q_{j+1}$
18 $\quad\quad \rho = \|q_{j+1}\|_2$
19 $\quad\quad q_{j+1} = q_{j+1} - Q_j c$
20 $\quad\quad \beta_j = \sqrt{\rho^2 - \sum_{i=1}^{j} c_i^2}$
21 $\quad$ end
22 $\quad q_{j+1} = q_{j+1} / \beta_j$
23 $\quad$ If $j < k$
24 $\quad\quad p_{j+1} = Aq_{j+1} - \beta_j p_j$
25 $\quad$ end
26 end

---

finite precision arithmetic, and for this reason we consider it as an estimation of the norm. Usually, these estimates are very accurate because the c_i coefficients are very small compared to $\|q_{j+1}\|_2^2$, that is, q_{j+1} and q'_{j+1} have roughly the same norm.

The first estimate (line 15) may be inaccurate if q_{j+1} is not fully orthogonal after the first orthogonalization. This does not represent a problem for the normalization stage, because in that case the criterion would force a reorthogonalization step and then a new norm estimation would be computed. Although the reorthogonalization criterion may seem less trustworthy, due to the use of estimates, the numerical experiments described in Section 6 reveal that this algorithm is as robust as Algorithm 5. In very exceptional cases, $\|q_{j+1}\|_2^2$ could be as small as $\sum_{i=1}^{j} c_i^2$, so that it is safer to discard the estimate and to compute the norm explicitly. This implementation detail is omitted from Algorithm 6 in order to maintain its readability.

The second major enhancement incorporated into Algorithm 6 is that the computation of α_j and the normalization of p_j are delayed until after the computation of $q_{j+1} = A^* p_j$ (these operations appear at the beginning of the loop in Algorithm 5). Thus, q_{j+1} must be corrected with this factor α_j in line 11,

$$q'_{j+1} = A^* p'_j = A^* \alpha_j^{-1} p_j = \alpha_j^{-1} A^* p_j = \alpha_j^{-1} q_{j+1}, \tag{5.4}$$

ETNA
Kent State University
http://etna.math.kent.edu

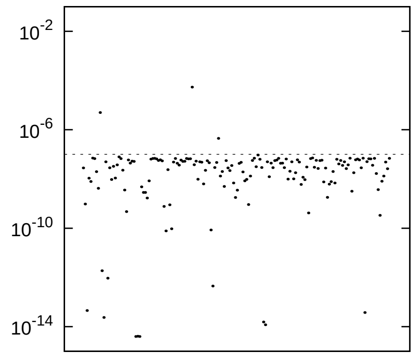

FIGURE 6.1. *Maximum relative error for each of the test matrices.*

where p'_j denotes the vector p_j after the normalization, and q'_{j+1} denotes the vector q_{j+1} after the correction. In lines 12 and 13, c and ρ are also corrected as

$$c' = Q_j^* q'_{j+1} = Q_j^* \alpha_j^{-1} q_{j+1} = \alpha_j^{-1} Q_j^* q_{j+1} = \alpha_j^{-1} c, \tag{5.5}$$

$$\rho' = \|q'_{j+1}\|_2 = \|\alpha_j^{-1} q_{j+1}\|_2 = \alpha_j^{-1} \|q_{j+1}\|_2 = \alpha_j^{-1} \rho. \tag{5.6}$$

In Algorithm 6, the communications associated with operations in lines 7-9 can be joined together in one multiple reduction message. In the same way, the operations in lines 17 and 18 can also be joined in one message. The rest of the operations, with the exception of the two matrix-vector products in lines 6 and 24, can be executed in parallel trivially (without communication). Therefore, the parallel implementation of Algorithm 6 needs only one (or two if reorthogonalization is needed) global synchronizations per iteration. This is a huge improvement over Algorithm 5, that has $j + 2$ synchronizations per iteration.

6. Numerical results. Algorithms 5 and 6 are not equivalent when using finite precision arithmetic. Therefore, their numerical behaviour must be analyzed. In order to check the accuracy of the computed singular values and vectors, the associated relative errors are computed explicitly after the finalization of the Lanczos process. If any of these values is greater than the required tolerance, then the bound used in the stopping criterion (2.12) is considered incorrect.

In this section, we perform an empirical test on a set of real-problem matrices, using the implementation referred to in Section 5, with standard double precision arithmetic. The analysis consists of measuring the relative error when computing the 10 largest singular values of non-Hermitian matrices from the Harwell-Boeing [12] and NEP [2] collections. These 165 matrices come from a variety of real applications. For this test, the solver is configured with tolerance equal to 10^{-7} and a maximum of 30 basis vectors. This relative error is computed as

$$\xi_i = \frac{\sqrt{\|Av_i - \sigma_i u_i\|_2^2 + \|A^* u_i - \sigma_i v_i\|_2^2}}{\sigma_i} \tag{6.1}$$

for every converged singular value σ_i and its associated vectors v_i and u_i.

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 7.1
Statistics and sequential execution times on Xeon cluster with the AF23560 matrix. Times are given in seconds and percentages are computed with respect to total time.

	Algorithm 5	Algorithm 6
Vector dot products	1,122	1,911
Matrix-vector products	116	116
Restarts	3	3
Total execution time	0.84 (100%)	1.22 (100%)
Vector dot products execution time	0.10 (12%)	0.22 (18%)
Vector AXPY execution time	0.36 (43%)	0.60 (49%)
Matrix-vector products execution time	0.34 (40%)	0.34 (28%)

TABLE 7.2
Statistics and sequential execution times on Xeon cluster with the PRE2 matrix. Times are given in seconds and percentages are computed with respect to total time.

	Algorithm 5	Algorithm 6
Vector dot products	3,225	5,108
Matrix-vector products	300	300
Restarts	9	9
Total execution time	86.58 (100%)	82.42 (100%)
Vector dot products execution time	12.61 (15%)	14.27 (17%)
Vector AXPY execution time	56.26 (65%)	49.68 (60%)
Matrix-vector products execution time	12.97 (15%)	12.96 (16%)

The results obtained running Algorithm 6 on these tests are shown in Fig. 6.1, where each dot corresponds to the maximum relative error for one matrix within the collections. Only in three cases (ARC130, WEST0156, and PORES_1), both Algorithm 6 and 5 produce some singular triplets with a residual larger than the tolerance. In these cases, there is a difference of six orders of magnitude between the largest and smallest computed singular values, and this causes instability in the one-sided variants. However, these large errors can be avoided simply by explicitly reorthogonalizing the left Lanczos vectors p_j at the end of the computation.

7. Performance analysis. In order to assess the parallel efficiency of the proposed algorithm (Algorithm 6) and compare it with the original one (Algorithm 5), several test cases were analyzed on two different computer platforms. In all these cases the solver was requested to compute 10 eigenvalues with tolerance set to 10^{-7}, using a maximum of 30 basis vectors.

On one hand, two square matrices arising from real applications were used for measuring the parallel speed-up. These matrices are AF23560 (order 23,560 with 460,598 non-zero elements) from the NEP [2] collection, and PRE2 (order 659,033 with 5,834,044 non-zero elements) from the University of Florida Sparse Matrix Collection [10]. The speed-up is calculated as the ratio of elapsed time with p processors to the fastest elapsed time with one processor. On the other hand, a synthetic test case is used for analyzing the scalability of the algorithm, measuring the scaled speed-up with variable problem size.

The first machine consists of 55 biprocessor nodes with Pentium Xeon processors at 2.8 GHz with 2 Gbytes of RAM, interconnected with an SCI network in a 2-D torus configuration. SLEPc was built with Intel compilers (version 10.1 with -O3 optimization level) and the Intel MKL version 8.1 library. Due to memory hardware contention problems, only one processor per node was used in the tests reported in this section. As shown in Tables 7.1

ETNA
Kent State University
http://etna.math.kent.edu

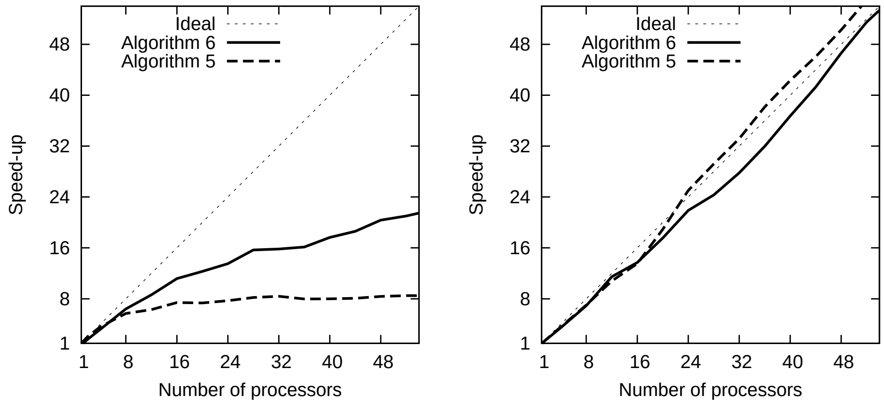

FIGURE 7.1. *Speed-up with AF23560 (left) and PRE2 (right) matrices on Xeon cluster.*

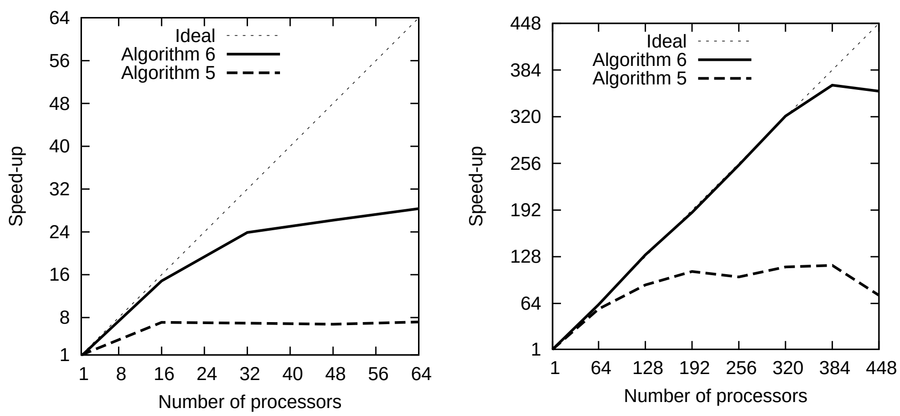

FIGURE 7.2. *Speed-up with AF23560 (left) and PRE2 (right) matrices on MareNostrum computer.*

and 7.2, both algorithms carry out the same number of matrix-vector products and restarts for both problems. Although Algorithm 6 performs more vector dot products than Algorithm 5 due to reorthogonalization, the sequential execution times are similar, with an advantage of the proposed algorithm for the larger problem. This is due to the fact that CGS with reorthogonalization exploits the memory hierarchy better than MGS. These tables also show the execution times for the different operations. The largest execution time corresponds to the vector AXPY operations that are used in the orthogonalization phase and in the computation of the Ritz vectors during restart (Eqs. 4.3 and 4.4). Regarding the benefits of explicitly storing A^* versus using MatMultTranspose, in these cases the gain is hardly perceptible, only about 0.08% reduction in the overall time.

As expected, Figure 7.1 shows that Algorithm 6 has better speed-up than Algorithm 5 with the AF23560 matrix. However, both algorithms show a good speed-up with the larger PRE2 matrix. In this case, the communication time is shadowed by the high computational cost of this problem and relative low number of processors.

These two tests were repeated on the MareNostrum computer in order to extend the

ETNA
Kent State University
http://etna.math.kent.edu

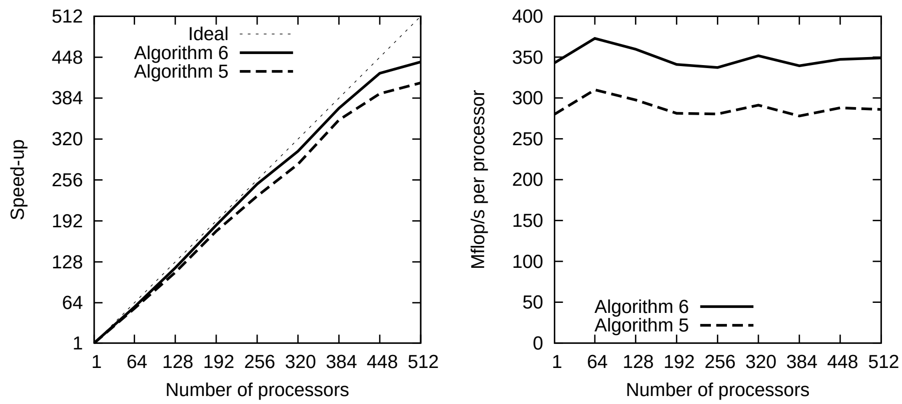

FIGURE 7.3. *Scaled speed-up (left) and MFlop/s rate per processor (right) with random tridiagonal matrix on MareNostrum computer.*

analysis to more processors. This computer is configured as a cluster of 2,560 JS21 blades interconnected with a Myrinet network. Each blade has two IBM PowerPC 970MP dual-core processors at 2.3 GHz and 2 Gbytes of main memory. The IBM XL C compiler with default optimization and the ESSL library were used to build SLEPc. The sequential behaviour of the two algorithms on this machine is similar to the one reported previously in Tables 7.1 and 7.2. Results with the AF23560 and PRE2 matrices (Fig. 7.2) show the clear advantage of Algorithm 6 over Algorithm 5 as the number of processors increases. The implementation described in this work obtained a quasi-linear performance improvement up to 384 processors with the PRE2 test problem.

In these fixed size problems, the work assigned to each processor gets smaller as the number of processors increases, thus limiting the performance with a large number of processors. To minimize this effect, the size of local data must be kept constant, that is, the matrix dimension must grow proportionally to the number of processors, p. For this analysis, a square non-symmetric tridiagonal matrix with random entries has been used, with a dimension of $100,000 \times p$. The scaled speed-up shown in the left plot of Fig. 7.3 is almost linear up to 448 processors, with a slight advantage for Algorithm 6. However, the proposed algorithm has significantly better throughput and gets closer to the machine's peak performance, as shown in the right plot of Fig. 7.3.

8. Discussion. In this paper, an optimized thick-restarted Lanczos bidiagonalization algorithm has been proposed in order to improve parallel efficiency in the context of singular value solvers. This algorithm is based on one-sided full reorthogonalization via iterated Classical Gram-Schmidt and its main goal is to reduce the number of synchronization points in their parallel implementation. The thick restart technique has proved to be effective in most cases, guaranteeing fast convergence with moderate memory requirements.

The performance results presented in Section 7 show that the proposed algorithm achieves good parallel efficiency in all the test cases analyzed, and scales well when increasing the number of processors.

The SLEPc implementation of the algorithm analyzed in this paper represents an efficient and robust way of computing a subset of the largest singular values, together with the associated singular vectors, of very large and sparse matrices in parallel computers. However, the standard Rayleigh-Ritz projection used by the solver is generally inappropriate for computing small singular values. Therefore, as a future work, it remains to implement also the possibility of performing a harmonic Ritz projection, as proposed in [1].

Acknowledgement. The authors thankfully acknowledge the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center (Centro Nacional de Supercomputación).

REFERENCES

[1] J. BAGLAMA AND L. REICHEL, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19–42.

[2] Z. BAI, D. DAY, J. DEMMEL, AND J. DONGARRA, *A test matrix collection for non-Hermitian eigenvalue problems (release 1.0)*, Technical Report CS-97-355, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1997. Available at http://math.nist.gov/MatrixMarket.

[3] Z. BAI, J. DEMMEL, J. DONGARRA, A. RUHE, AND H. VAN DER VORST, eds., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[4] S. BALAY, K. BUSCHELMAN, V. EIJKHOUT, W. D. GROPP, D. KAUSHIK, M. KNEPLEY, L. C. MCINNES, B. F. SMITH, AND H. ZHANG, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 2.3.3, Argonne National Laboratory, 2007. Available at http://www.mcs.anl.gov/petsc/petsc-as.

[5] M. W. BERRY, Z. DRMAČ, AND E. R. JESSUP, *Matrices, vector spaces, and information retrieval*, SIAM Rev., 41 (1999), pp. 335–362.

[6] Å. BJÖRCK, E. GRIMME, AND P. VAN DOOREN, *An implicit shift bidiagonalization algorithm for ill-posed systems*, BIT, 34 (1994), pp. 510–534.

[7] A. COOPER, M. SZULARZ, AND J. WESTON, *External selective orthogonalization for the Lanczos algorithm in distributed memory environments*, Parallel Comput., 27 (2001), pp. 913–923.

[8] J. CULLUM, R. A. WILLOUGHBY, AND M. LAKE, *A Lanczos algorithm for computing singular values and vectors of large matrices*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 197–215.

[9] J. W. DANIEL, W. B. GRAGG, L. KAUFMAN, AND G. W. STEWART, *Reorthogonalization and stable algorithms for updating the Gram–Schmidt QR factorization*, Math. Comp., 30 (1976), pp. 772–795.

[10] T. DAVIS, *University of Florida Sparse Matrix Collection*. NA Digest, 1992. Available at http://www.cise.ufl.edu/research/sparse/matrices.

[11] J. W. DEMMEL AND W. KAHAN, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873–912.

[12] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[13] L. ELDÉN AND E. SJÖSTRÖM, *Fast computation of the principal singular vectors of Toeplitz matrices arising in exponential data modelling*, Signal Processing, 50 (1996), pp. 151–164.

[14] J. FRANK AND C. VUIK, *Parallel implementation of a multiblock method with approximate subdomain solution*, App. Numer. Math., 30 (1999), pp. 403–423.

[15] G. H. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal., Ser. B, 2 (1965), pp. 205–224.

[16] G. H. GOLUB, F. T. LUK, AND M. L. OVERTON, *A block Lánczos method for computing the singular values of corresponding singular vectors of a matrix*, ACM Trans. Math. Software, 7 (1981), pp. 149–169.

[17] M. HANKE, *On Lánczos based methods for the regularization of discrete ill-posed problems*, BIT, 41 (2001), pp. 1008–1018.

[18] V. HERNÁNDEZ, J. E. ROMÁN, AND A. TOMÁS, *Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement*, Parallel Comput., 33 (2007), pp. 521–540.

[19] V. HERNÁNDEZ, J. E. ROMÁN, A. TOMÁS, AND V. VIDAL, *SLEPc users manual*, Tech. Rep. DSIC-II/24/02 - Revision 2.3.3, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007. Available at http://www.grycap.upv.es/slepc.

[20] V. HERNÁNDEZ, J. E. ROMÁN, AND V. VIDAL, *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Trans. Math. Software, 31 (2005), pp. 351–362.

[21] Z. JIA AND D. NIU, *An implicitly restarted refined bidiagonalization Lanczos method for computing a partial singular value decomposition*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 246–265.

[22] S. K. KIM AND A. T. CHRONOPOULOS, *An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices*, Int. J. Supercomp. Appl., 6 (1992), pp. 98–111.

[23] E. KOKIOPOULOU, C. BEKAS, AND E. GALLOPOULOS, *Computing smallest singular triplets with implicitly restarted Lanczos bidiagonalization*, App. Numer. Math., 49 (2004), pp. 39–61.

[24] L. KOMZSIK, *The Lanczos Method: Evolution and Application*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2003.

[25] R. M. LARSEN, *Lanczos bidiagonalization with partial reorthogonalization*, Tech. Rep. PB-537, Department of Computer Science, University of Aarhus, Aarhus, Denmark, 1998.
Available at http://www.daimi.au.dk/PB/537.

[26] ———, *Combining implicit restart and partial reorthogonalization in Lanczos bidiagonalization*, Tech. Rep., SCCM, Stanford University, 2001.
Available at http://soi.stanford.edu/~rmunk/PROPACK.

[27] B. N. PARLETT, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Englewood Cliffs, NJ, 1980. Reissued with revisions by SIAM, Philadelphia, 1998.

[28] L. REICHEL AND W. B. GRAGG, *FORTRAN subroutines for updating the QR decomposition*, ACM Trans. Math. Software, 16 (1990), pp. 369–377.

[29] Y. SAAD, *Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms*, John Wiley and Sons, New York, 1992.

[30] H. D. SIMON, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.

[31] H. D. SIMON AND H. ZHA, *Low-rank matrix approximation using the Lanczos bidiagonalization process with applications*, SIAM J. Sci. Comput., 21 (2000), pp. 2257–2274.

[32] K. WU AND H. SIMON, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.